Performance Planning and Troubleshooting for CMS

Table of Contents

Production will insert TOC here and it will be based on Heading 4.

Introduction

Getting the best performance from your Microsoft Content Management Server 2002 (MCMS) Web site is an important strategy for getting the most out of your investment. Good performance not only makes for a better experience from your Web site, but also reduces hardware costs by using your servers as efficiently as possible.

This white paper describes the various ways in which MCMS can be optimized for performance. The first sections describe the most important architectural features of MCMS to understand for performance optimization. The next sections describe performance optimization strategies based on your role: Web developer, IT administrator, or site architect. Finally, the troubleshooting section demonstrates how to diagnose and fix an underperforming MCMS site.

Who should read this white paper?

This white paper is aimed at site architects, IT administrators, and Web developers who are using or planning an MCMS Web site. Site architects will learn how to structure the site and network, identify bottlenecks, and scale the infrastructure to accommodate traffic demands. IT administrators will learn how to optimize performance of the MCMS servers and related applications like SQL Server and Internet Information Server. Web developers will learn how to build templates and design navigation for best performance.

When should you consult this white paper?

This white paper is a valuable resource when planning your Web site and building a trial version of the site. It sets out guidelines that will help you design your trial site and its pages with performance in mind. Through the use of good design practice early, you can avoid a costly redesign later when you take your site to production. Designing for performance from the beginning is also important if you plan to do transaction cost analysis on your site using the Transaction Cost Analysis for MCMS 2002 <link> white paper. Transaction cost analysis (TCA) can help you determine the precise hardware your Web site requires to meet its performance targets for the expected traffic load. To do TCA effectively, you must have the major structure of the Web site in place, including the site and page design.

See these documents for more information on developing Web sites with MCMS:

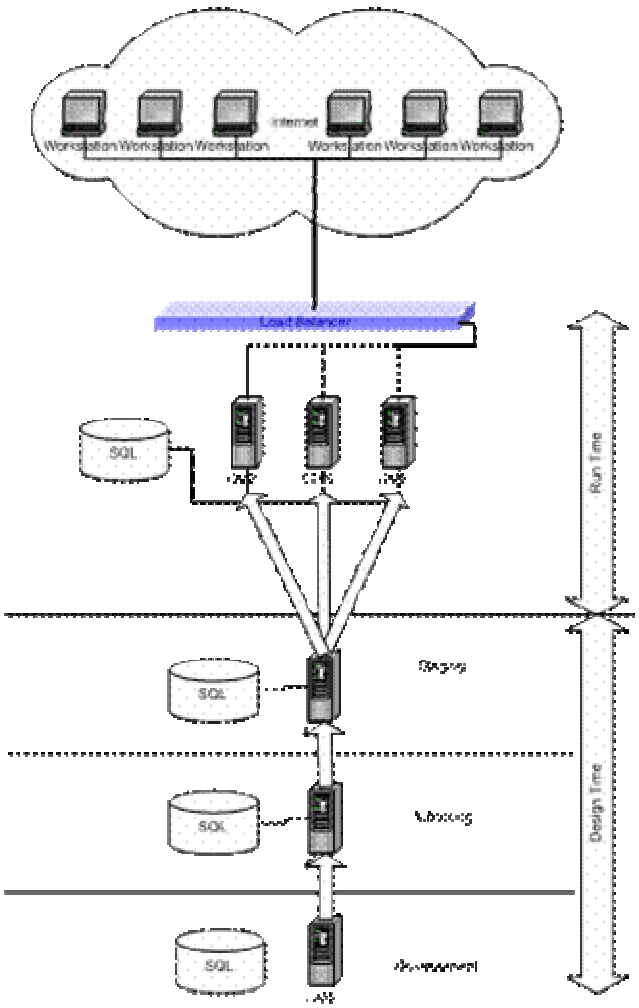
MCMS Product Documentation

WoodgroveNet Sample Site

MCMS Product Overview

The MCMS site structure

A well-structured MCMS environment is tiered, allowing for the separation of development, authoring, staging and testing, and production. Each tier consists of at least one MCMS server and a SQL Server. This tiered architecture reflects the organization or process in which an MCMS Web site is designed, assembled, and published to the Web.



In the development tier, Web developers create templates and program navigation bars.

In the authoring tier, business users use the templates to create content and take it through an approval process to publish the content to the site. Approved content is assembled in the authoring server.

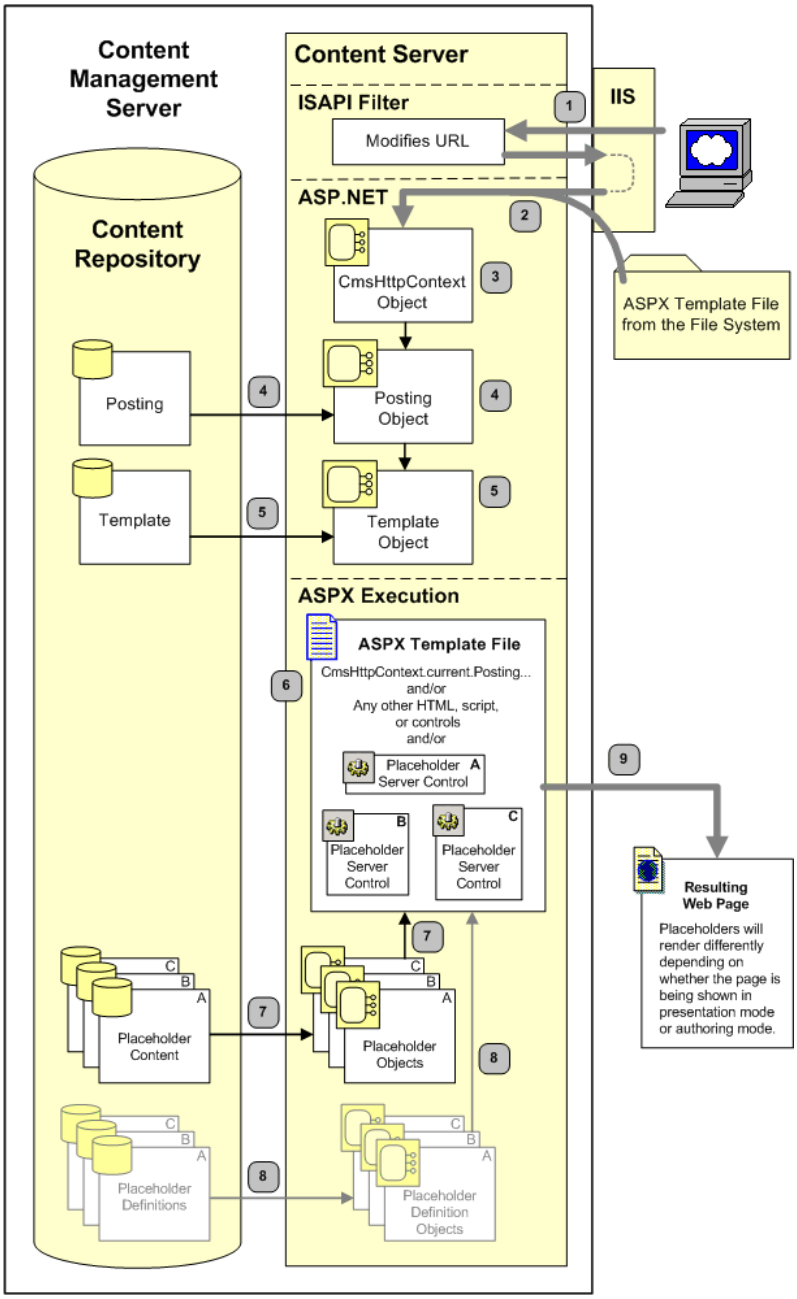
The staging tier, if necessary, allows content to be tested before publishing. This stage is also used for load testing.

The production tier serves incoming page requests. Generally, the production tier consists of several MCMS servers supported by a SQL Server. The production tier may even be a web farm with multiple clusters of MCMS servers, with each cluster supported by its own SQL server. A load balancer handles incoming requests and distributes them among the MCSM servers.

Serving up a page

One of the strengths of MCMS is that it responds to page requests by dynamically assembling the Web pages it serves. That is, it is not limited to a list of static Web pages built in advance. Instead, it can respond to each request by assembling the required Web page, which means that Web sites can be filled with dynamic content targeted at individual users.

However, this flexibility comes at a certain cost: it can take more time to assemble a new page than to simply serve up an existing page. MCMS compensates for this by using a sophisticated system of memory caching that allows it to have the performance of a static page server, while maintaining the flexibility of dynamic page assembly. Many of the strategies for optimizing MCMS performance revolve around caching and page assembly, so it is worthwhile to review how an MCMS server actually serves up a page. The diagram at right shows the steps that a server goes through when it receives a request for a URL.



1. A page request arrives through Microsoft Internet Information Services (IIS) and is trapped by the MCMS Internet Server API (ISAPI) filter, which takes apart the URL and finds the corresponding posting. It follows the posting to the template file, and then constructs a new URL using the ASPX template file. The new URL contains several query string parameters, including the globally unique identifier (GUID) of the posting. This transformed URL is then passed back to IIS, where processing continues.

2. IIS passes the transformed URL into the ASP.NET DLL, where a variety of processing is performed. This includes the required authentication processing, caching, when enabled, and processing that establishes the MCMS context in which the ASPX page will be executed. Authentication and cache processing are not shown here for the sake of simplicity.
3. MCMS instantiates and initializes a CmsHttpContext object that serves as the MCMS object model entry point for the ASPX template file that is executed in later steps.
4. Initialization continues with the instantiation and initialization of the Posting object that corresponds to the requested page. Recall that the GUID of the current posting was passed as a query string parameter in the transformed URL. Information about the posting is read from the Content Repository.
5. Initialization continues with the initiation and initialization of the Template object for the template upon which the requested page is based. Information about the template is read from the Content Repository.
6. Execution of the ASPX template file begins. The MCMS context established in the previous steps is accessible to code in the ASPX template file using the current property of the **CmsHttpContext** object as in the following example:

```
CmsHttpContext.Current.Posting
```

The ASPX template file typically contains placeholder controls, but it can also contain any other HTML, script, and/or non-MCMS controls, as appropriate.

7. Assuming that the template file contains one or more placeholder controls, when these controls execute, they instantiate and initialize Placeholder objects that contain the actual content to be displayed in the controls. Information about the Placeholder object(s), including the actual placeholder content, is read from the Content Repository.
8. Making the same assumption about the presence of placeholder controls in the template file, these controls instantiate and initialize the **PlaceholderDefinition** objects that correspond to the **Placeholder** objects. These objects contain configuration and constraint information about the content to be displayed in the controls. Information about the **PlaceholderDefinition** object(s) is read from the Content Repository. Note that these objects appear dimmed in the figure, indicating that they do not play much of a role in the process of displaying a requested page (as opposed to what occurs when the content on a page is authored).
9. During the course of the execution of the template page, the HTML that constitutes the resulting Web page is emitted and sent to the user.

Basic principles in MCMS performance

The first principal for good performance is to ensure that your MCMS servers have sufficient memory to store the working set of pages in output or fragment cache. Because cached pages always serve fastest, increasing the memory available to the output and fragment caches is always the first path to improving performance.

However, pages may not always be present in output or fragment cache: the first time a page is requested, it must be assembled from templates, placeholder data, and resources. Factors such as access patterns and update frequency can also reduce cache

effectiveness. Therefore, the next route to improving performance is to speed the assembly of pages. The best way to speed page assembly is to ensure that there is sufficient memory available for the node cache, which stores the MCMS objects, such as templates and placeholder data that are required to assemble the pages.

Next, processor capacity is important. Ensure that there is enough CPU capacity available to efficiently assemble pages using data from node cache, disk cache, and the SQL server database.

The size of the disk cache, which stores resources, can also speed page assembly. A sufficiently large disk cache can reduce or eliminate the need to go to the SQL Server database for resources.

Finally, you can improve performance by reducing the resource and processing demands that your page and site design pose. Template design, navigation, and site architecture considerations all play a role. Some template designs will increase the processing burden. A too-complex site design can do the same. Certain kinds of content, or content targeted at specific users will also slow down performance. Because every site offers different content for different users, the performance challenges will be different for each site.

The purpose of performance optimization is to find the best configurations and structures to allow your site to achieve the best performance possible.

Setting performance targets

For the purposes of planning and designing a Web site, performance targets are generally expressed in terms of performance metrics: pages per second, transactions per second, guest sessions per second, and so forth. However, determining just what those targets will be can often be a bigger problem than choosing a counter to measure them. How many users will there be? What response time do they find acceptable? What kind of information will they access?

For projects involving the migration of an existing site to MCMS 2002, the problem is relatively easy because traffic measures can be taken of the old site. For new, growing, or substantially redesigned sites, the problem is more difficult because historical data is unavailable or inapplicable. For every site, however, performance targets eventually come from the business requirements of the site.

Business requirements can be vague and inexact, but they are an essential starting point for capacity and performance planning. The challenge is to translate business requirements into measurable numbers.

There is no quick formula to translate business requirements into throughput numbers. However, consider the following questions when formulating your performance targets:

- What will be the traffic patterns for your site? That is, will your site have constant, steady traffic, or will there be times when traffic will spike? If your site offers movie trailers, for instance, you can expect a spike in traffic when a new trailer comes out. Therefore, you should design your Web site for these peak periods, as opposed to the quiet periods in between.
- What is the longest latency you can tolerate? Business requirements will often call for instant response time, but this is not generally possible or practical. Instead, set

an upper limit for the average wait between request and response, in seconds. As a general rule, latency times above 5 seconds will start to discourage users.

- What are your availability requirements? If your site must be running 99.99 percent of the time, you will have to design a site with redundancy using clustered web servers or that has high failover capabilities. This may translate into multiple servers or multiple clusters of servers.
- Will your content be highly personalized? Highly targeted Web content will perform differently from generic content.

There are many other questions to consider when determining the performance targets for a site. For more information, see the link to the user profile white paper in the "Testing your site" section below.

Site development

Site development refers to the designing and programming of Web page templates and navigation, and is usually performed by ASP developers. Web developers can have a large impact on the performance of a Web site. Templates that have been optimized for performance will produce Web pages that can be quickly assembled. Efficiently coded navigation bars will render quickly and not place too much demand on the server.

ASP versus ASP.NET

One of the major changes in MCMS 2002 from MCMS 2001 is support for ASP.NET. MCMS 2001 supported only ASP page development. ASP.NET offers several advantages over ASP:

- ASP.NET comes with its own caching mechanism, referred to as output cache. Programmers can have control over what is cached and how long pages or parts of the page are cached to optimize the performance.
- You can design and program templates in MS Visual Studio .NET, which provides a WYSIWYG environment, makes adding controls easy, and has a fully supported debugging environment.
- You can program templates in Visual Basic .NET, C#, or JScript.
- ASP.NET pages can take advantage of the .NET Framework. For more information on the .NET Framework, see the Microsoft .NET Framework Web site.

All these features will make it easier for you to optimize your ASP.NET pages for performance. Therefore, new pages should always be developed using ASP.NET instead of the older ASP.

Placeholders and template design

A template is an MCMS object that is stored in the Content Repository and serves as the design for a particular set of pages. These pages are said to be based on that template. Templates encapsulate the placeholder and custom property definitions, and identify the template file (which contains the executable code and controls) for their pages.

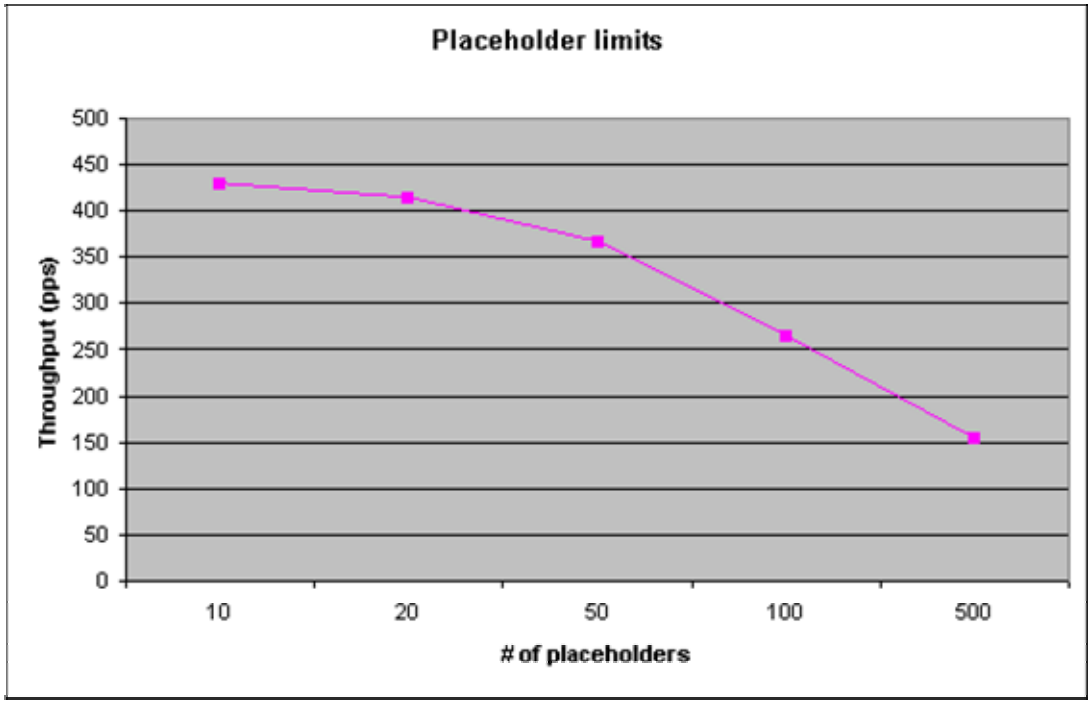
The key feature of a template is the placeholder. Placeholders are the areas on the template where text, images, or rich media may be placed. Placeholders are what make a template a template, but if mismanaged they can degrade the performance of the site.

There are two basic strategies for managing placeholders: limit their number, and limit their size.

Limit the number of placeholders

The more placeholders there are on a template, the more objects and data have to be retrieved to assemble the page, and the more processing a page requires. In some ways, you can trade off the number of placeholders against the number of templates. It may be possible, for instance, to have a single main template from which all pages are derived through the use of complicated design of the templates with multiple placeholders. However, this arrangement will place the maximum burden on the server.

As you can see from the chart below, the page-per-second throughput tapers off when more placeholders are introduced into the template.



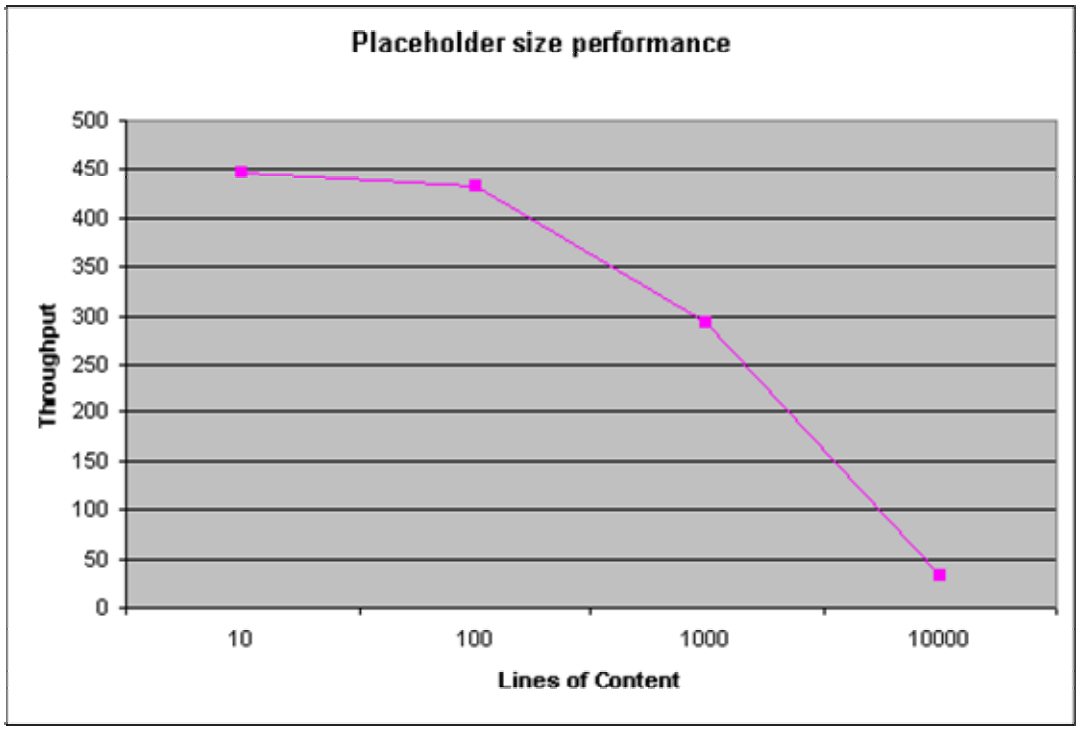
The slope of the curve will vary depending on size and type of placeholder you add. As a general rule, limit the number of placeholders on a template to fewer than 100. For best performance, consider limiting the number of placeholders to 30 or less to avoid a decline in throughput.

Limit the size of data in placeholders

As noted earlier, the size of the data contained by the placeholders in a template also has a large impact on performance. As images grow bigger and the number of lines of content of a template increases, performance drops off dramatically.

Again, you can trade off certain parameters in your template design to get the best performance. For instance, if a placeholder has a large amount of content (1000 lines or more) you can split up that content into multiple placeholders. While this will increase the total number of placeholders, the individual placeholders will be smaller. You will have to find the right balance between these two parameters to achieve the best performance.

Of course, the size of other types of placeholders should always be restricted as much as possible, particularly images and rich media such as animation and video. But these size/performance trade-offs must be made with any Web site. The chart below shows how throughput falls off as the size of placeholder data increases. (Note that the horizontal axis is logarithmic to make this fall-off more evident.)



Navigation

Navigation controls, like tables of contents or navigation trees, have a large impact on MCMS performance because they are generally used on every page. A badly designed navigation control can be one of the reasons behind high latency issues in your final pages. Therefore, a little time spent optimizing your navigation code can have an enormous impact on overall Web site performance.

There are two aspects to a well-designed navigation control: rendering only what is necessary and making the site-level navigation as generic as possible.

Tree-style navigation controls can be very useful for large sites with highly organized content, like technical support sites. However, tree navigation controls can require significant time if the entire tree is computed all at once. A better solution is to load and compute only that part of the tree required for that particular page. As the user navigates through the content, the remaining part of the tree required can be identified

or computed and loaded as necessary. In this way, computing the tree control as the user moves through the navigational controls eliminates the wait caused by computing the entire tree at the startup. It can also reduce the overall loading time as the parts of the tree that are not used will not have to be computed and loaded. Preloading the entire tree may be easier for a small site; however this technique does not scale and can cause serious performance degradation in deep sites. The incremental loading technique reduces initial overhead.

Another key strategy to minimizing the performance cost of navigation is to use the same navigation controls site-wide. Although this can result in larger tree controls, it allows you to take advantage of the output cache or fragment cache, which can outweigh the increase in size of the control. When a common navigation control is used on many or all the pages on a site, it can be kept in the cache to speed up page assembly. The more the navigation is customized—for instance, on a user-level basis—the less cost-effective it will be to cache it. Try to design your site around very little or no targeted customization in the navigation tree.

Authentication

Authentication is required when the site has security concerns or when content is personalized and targeted to particular users. Targeting content requires that users be identified and authenticated, and this authentication carries a certain performance burden.

The overall performance of MCMS can depend on the type of authentication used. Web sites configured for guest access—that is, no authentication—have been shown to perform the best. Sites that require users to log in when they use the site—manual authentication—perform up to 30 percent slower than those that allow users access as guests. You should, of course, consider these results when building your site. A more radical solution would be to keep targeted content and generic content on different sites entirely.

If you have a guest-access-only site, you can improve the performance of the system by removing the MCMS Authorization module. In your ASP.NET sites, this can be done by removing the authorization module from the Web.config file. See "Configuring Internet Sites" for more information.

The chart below shows that performance gains of up to ten percent may be obtained by removing the authorization module. As you can see, the performance improvement is significant only on servers with a low number of processors and decreases as the number of processors increases.

Caution: You should only remove the authorization module in sites that contain content for guest access only. If the site contains sensitive data meant only for registered (non-guest) users, this data may become accessible to guest users if the authorization module is removed.

Network and hardware management

Content Management Server has been built to run high quality, high performance, high volume Web sites. To achieve that performance, however, it is crucial that the machines that run MCMS have enough resources to support the traffic, and that these

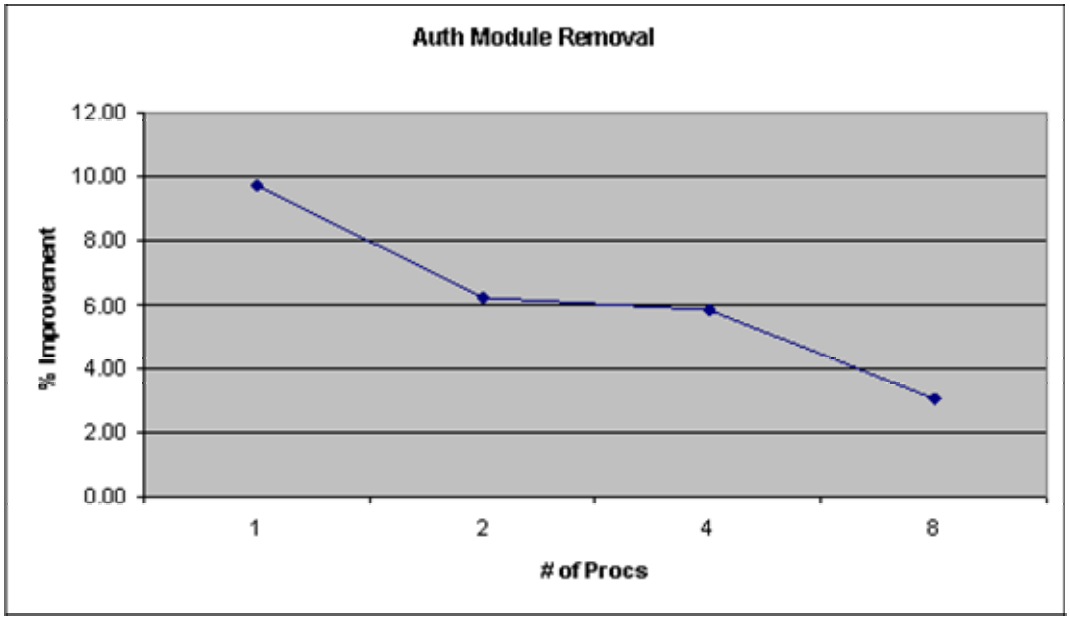
resources are optimally used. Server optimization will allow you to meet your performance goals without over-investing in hardware.

There are many aspects to server management. Cache management can have the biggest immediate impact on performance, but improper disk management can also slow down performance. Tasks like content deployments and routine housekeeping, such as background processing, can leave the server underperforming periodically, and must be minimized or scheduled for off-peak hours. Network latency from various sources will always affect performance. If your total system resources are not enough to handle the traffic demands, you may have to scale up your servers or scale out your server farm. Finally, MCMS relies on SQL Server 2000 and Internet Information Server to store and serve pages; inappropriate configurations in these applications can reduce your site's performance.

These aspects of network and hardware management are covered in the sections below.

Cache balancing

As described in the "Serving up a page" section above, there are four different kinds of cache in Content Management Server: output cache, fragment cache, node cache, and disk cache. Three of these caches—output, fragment, and node—reside in memory. Disk cache obtains resources from MCMS resource galleries and stores them temporarily on the hard disk.



Because the output, fragment, and node caches will all compete for memory resources, you must balance their resource demands to optimize performance. Of these, node cache can be configured using the Server Configuration Application (SCA), whereas output cache can be manipulated using cache control features or ASP.net. Each of these cache types is described below.

ASP.NET output cache

Output is the first cache that the ASP.net checks for incoming MCMS page requests. The output cache is stored in RAM, and the content it stores can be served up at a rate of hundreds of pages per second.

This cache is actually part of the ASP.NET Framework and is managed by the Framework. Therefore, output cache is only available for ASP.NET pages, and not ASP pages. It stores static chunks of HTML, either fragments of pages or whole pages, and is usually implemented on template files or user controls on which postings are based. You must enable output caching for each template and control you wish to store in the cache. For details on how to enable output caching, consult the MCMS SDK documentation and ASP.NET documentation.

Because output caching increases performance so dramatically, all MCMS sites using ASP.NET should make use of this cache where appropriate. Increasing the amount of content that is served out of this cache is often the single biggest performance enhancement that you can make to an MCMS site.

Note that postings and objects which have been output cache-enabled are added to the output cache the first time they are requested. They remain in cache until their duration expires (see the "Caching concerns" section below) or the cache is flushed (see the "Content updates" section below).

Fragment cache

The fragment cache is similar to output cache in that MCMS uses it to store page components or whole pages in RAM. The main difference between output cache and fragment cache is that fragment cache is not part of the ASP.NET Framework, but is created and managed directly by MCMS. This means that the fragment cache can be used to cache only ASP pages.

The other major difference between output caching and fragment caching is in the way the caches are built and flushed. While the output cache is built up as pages and resources are requested, the entire fragment cache is loaded into memory at once. And while the output cache gets flushed when content is updated, the fragment cache does not.

As a rule, fragment caching is used only with "guest" content. Fragment caching user-specific content requires a great deal of care to ensure that sensitive information is not accidentally revealed.

CMS node cache

If a requested page is not available in the output or fragment caches, MCMS will have to assemble the page from its constituent parts: templates, resources, and placeholder data. MCMS uses node cache and disk cache (described below) to assemble pages. Node cache stores internal MCSM data such as channels, templates, resource metadata, template galleries, and resource galleries, as well as the placeholder data themselves. The internal MCMS data provides the roadmap to allow MCMS to locate the requested resources in disk cache.

Because the node cache plays such a central role in page assembly, increasing the amount of memory available to node cache is an important part of improving MCMS performance. You can set the size of this cache through the SCA.

Disk cache

The disk cache resides on the MCMS Servers Disk and essentially contains a local copy of the resources stored in MCMS repository on the SQL Server. Therefore the use of the disk cache allows MCMS to retrieve resources without resorting to a network call to the SQL Server. To completely eliminate frequent calls to the database, you should increase the size of the disk cache so that it can contain most, if not all, of your MCMS managed resources; you can do this through the SCA.

Performance Tip: If the server's disk cache directory is installed on the same drive as Microsoft Windows, consider moving the Internet Information Services (IIS) log files from the default location in the Windows directory to another drive. This prevents log files from accumulating and causing problems with the ability to cache data or move the disk cache directory to another drive.

Caching concerns

There are several factors that can affect caching, including the following:

- **The amount of memory on your MCMS and SQL servers.** Windows 2000 Advanced Servers can accommodate up to 4GB of RAM; increasing memory can be the fastest, easiest, and least expensive way to increase performance.
- **The distribution of page hits (what percentage of all hits go to the most frequently hit pages).** Usage profiles and site traffic analysis can help you determine which pages generate the most traffic. For best performance, ensure that output cache can store the Web pages that account for 90 percent or more of total requests.
- **The rate at which content needs to be updated (and the output and node caches need to be flushed).** For more information on content updates, see the following section.

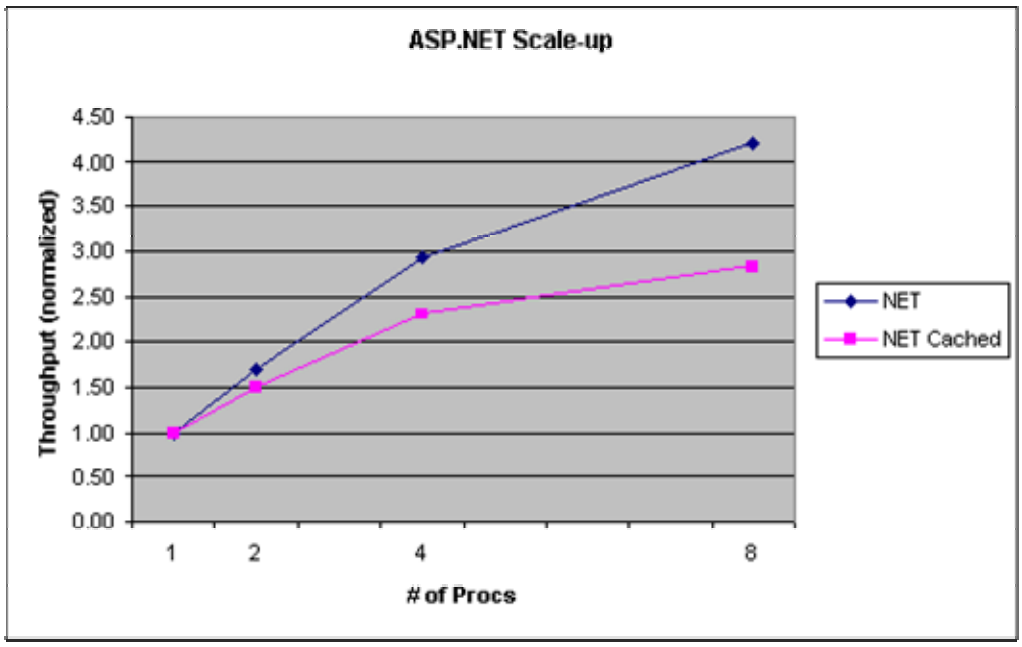
Those pages that cannot be served from output and fragment cache must be assembled by the node and disk caches. Ideally, MCSM servers should only access the SQL database during content updates.

Scaling up

As noted earlier, assembling MCMS pages is a processor-intense activity. Because in a well-tuned system the processor will be the limiting factor, you can improve throughput for an MCMS server by adding more CPUs.

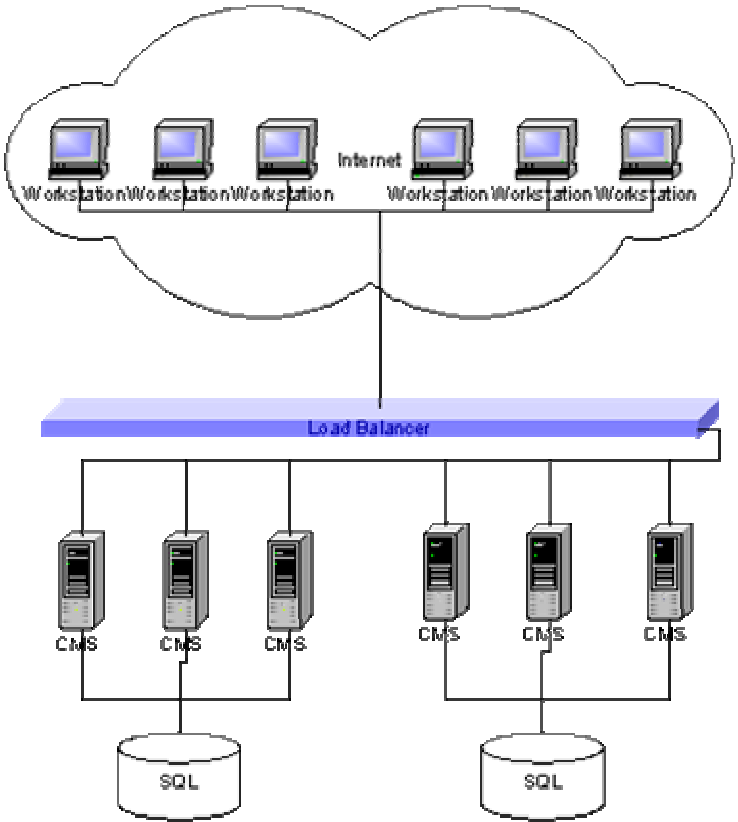
However, please note that doubling the number of CPUs will not double the throughput of the server. As you can see from the results below, throughput does not increase linearly as CPUs are added. In many instances, it is better to cluster two four-processor machines together rather than using a single eight-processor machine. (See the "Scaling out" section, below.)

Note that this chart shows the normalized throughput for .NET pages and cached .NET pages. That is, the throughput numbers have been adjusted so that for both the cached and uncached results, the throughput value for one processor is 1. However, in reality cached pages serve much faster; for the results this chart is based on, the throughput for cached pages was over 7 times faster than for uncached pages for a single processor. For 8 processors, the difference was almost 5 times.



Scaling out

Scaling out refers to increasing the number of MCMS servers or server clusters available to respond to large numbers of page requests. Scaling out also provides redundancy to handle load if one or more servers goes down.

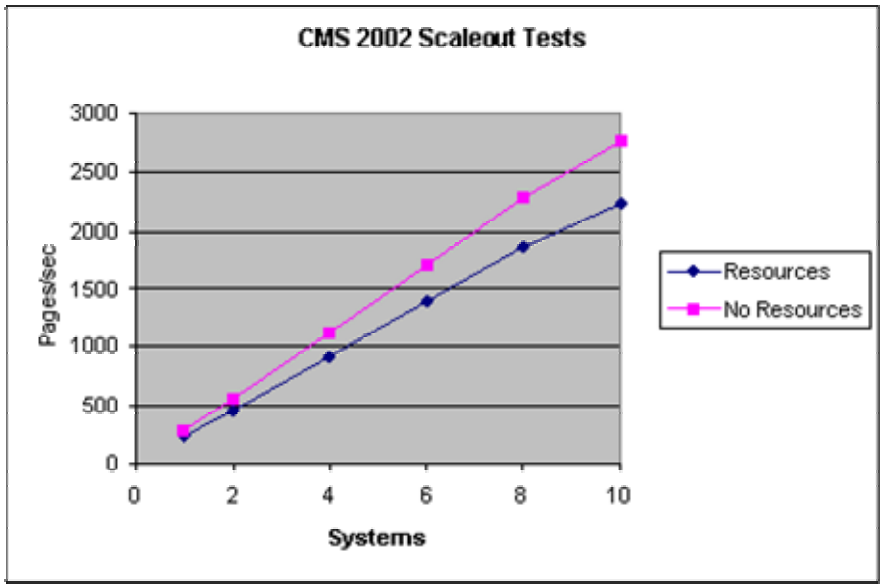


Scaling out at the MCMS level is achieved by load balancing across one or more MCMS clusters. You can use either a hardware load balancer (HLB) or a software load balancer such as Windows Network Load Balancer (NLB).

Whether you use HLB or NLB is your choice, but they may have different performance characteristics. For most sites, a software load balancer may be adequate. However, for very

high throughput sites, you may want to evaluate using HLB.

We recommend that for extremely high performance sites, such as those serving out greater than 300 pages per second across a cluster with four or more MCMS servers, a hardware load balancer be used. However, most sites do not reach such high throughput levels, and when the Web servers are used in a load balanced configuration, the costs distribute evenly. The diagram below shows the throughput scales linearly as you increase the number of machines within the configuration. No one Web server runs in a more expensive mode to handle the work even as the load increases.



Content updates

Posting new material to your Web site is, of course, one of the most important functions. But content updates can pose a problem for any content management product because they require sometimes extensive updating of a large pool of information and ensuring that outdated content is not still offered by the Web site.

When new content is posted to Content Management Server, one of the main effects is to flush those parts of the output and node caches that have been updated. This ensures that outdated content will not remain in cache and get served up accidentally. However, flushing the caches comes with a temporary but substantial performance hit, as new pages cannot be served up out of cache until the cache is rebuilt. Cache is rebuilt from the updated disk cache as the new pages are requested; therefore the new pages take longer to serve because they must be assembled first.

The solution to this problem is to batch content updates as much as possible, instead of posting new content to the site as it comes available. The frequency of new content batches will depend on your business requirements for the site: how urgent is it that the new content be posted?

Background processing

Background processing removes expired content; that is, content which is past its publication interval. It also removes any dangling resource data that is already deleted. Background processing refers to the task of deleting old and expired pages from the database to reduce database growth and increase efficiency.

Background processing has been improved in MCMS 2002 to run at the SQL layer instead of at the MCMS server, which means that your MCMS server is not affected during processing. This means that any cached pages, whether in the output and fragment caches or the data in MCMS node cache, will continue to be served out.

However, there is still a potential performance hit for MCMS. If the item is not in the cache, the server must go back to the database to retrieve the page content. Database performance will decline during background processing. Therefore, as a general rule you should schedule background processing for non-peak times to reduce the chance that it will affect your site performance.

Network latency

Network latency, the delay in retrieving information from another machine due to network hops, or bandwidth, can impact the performance of MCMS. In general, ensure that the network latency to any external connections such as SQL Servers and Active Directory Servers is kept to a minimum.

One way to minimize network latency is to ensure that the Web server is placed on the same switch as the database servers. Other network management parameters may be adjusted as well; however, as network management is beyond the scope of this paper, they will not be discussed here.

Caution: If you must change these network parameters in testing or in a live site, change only one setting at a time and compare the new results carefully with the old. Careless changes to the parameters will make administration and management difficult.

Managing IIS

Content Management Server incorporates Internet Information Server to serve up Web pages. IIS can therefore be the site of some performance tuning.

To increase the performance of IIS when running a runtime only site, you can remove the Resolution HTML packager ISAPI filter in the rehtmlpackager.dll file from the MCMS server. This filter is used to display differences between versions of a document in MCMS. Therefore, removal of this filter will not affect MCMS functionality in serving out pages on a read-only site. You can realize a performance improvement of approximately 10 percent after removal of this filter.

To remove the ISAPI filter, open IIS and go to Web Sites. Right click Properties -> ISAPI filters and select Resolution HTML Packager ISAPI filter. Then click Remove.

There are several other ways you can tune IIS to obtain the best performance:

- You can set the IIS process isolation level to improve throughput.

- You can set the IIS Request/day slider on the IIS Server Properties window to the amount of traffic expected.
- You can configure IIS to be optimized for performance.

These options are all available through Internet Services Manager. Please consult the documentation for IIS for more information.

Managing SQL Server

For SQL Servers, typically the most important tuning option is setting up the physical disk subsystem. For optimal performance, the databases should be separated from their transaction logs on different physical drives to prevent resource conflicts. All of the databases, the transaction logs, and the TempDB should be set up so that each individual disk subsystem is not the bottleneck. You should carefully correlate the disk costs with the transactions in order to plan for increased disk requirements.

The SQL disk or CPU is seldom a bottleneck with read-only sites. However read/write sites with a lot of authoring will often cause the SQL server to become the bottleneck. See the "Design-time performance" section below.

SQL Server also takes advantage of large amounts of physical memory, so the amount of RAM available should be weighted against the working set of the database. During run-time, the network IO and the processing load on the SQL Server is a direct function of the number of front-end servers accessing the SQL Server database as well as the profile of the load.

Should none of the MCMS caches (output or fragment, node, or disk cache) contain the requested content, a request will be sent the SQL Server. By default SQL tries to store frequently accessed data in RAM. This cache grows to consume all available memory on the SQL server without using virtual memory. As such, SQL Server should have sufficient memory for best performance. This will limit disk access as much as possible. For more information about SQL Server performance tuning, go to **http://go.microsoft.com/fwlink/?LinkId=9512.**

Design-time performance

The usage pattern of a typical web site consists of a large number of subscribers viewing web pages from the site while a small number of authors update content rather infrequently. Content Management Server is primarily designed to handle these typical web site usage patterns, consisting of large number of read requests with a smaller number of write operations.

Nevertheless, the performance can be affected due to database contentions between read operations from Web site user requests and write operations from the updates to the content. To avoid performance impacts, you should separate the site into a production tier consisting primarily of read operations and an authoring tier consisting primarily of updates and search operations.

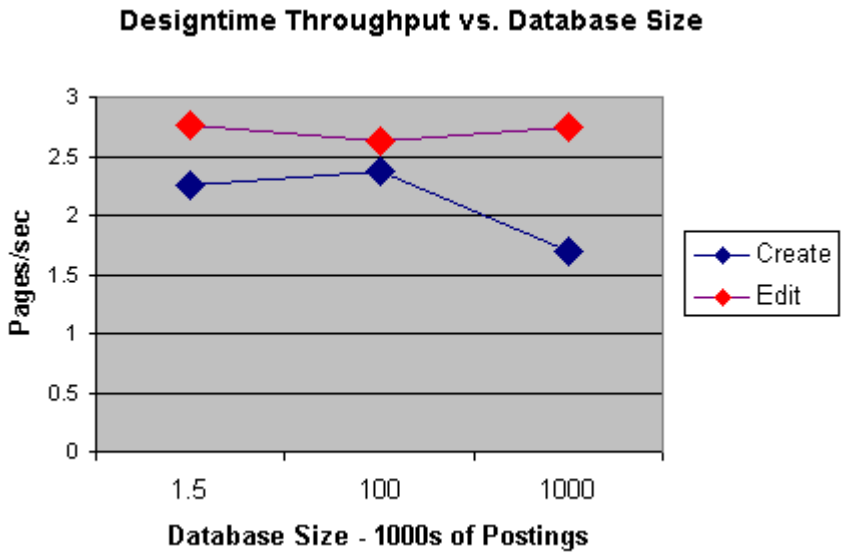
MCMS is designed to handle the design-time throughput of a typical organization. It can handle a reasonably large number of authors (25-100) each updating small number of postings (~10) per day. However, the performance of MCMS will suffer under a high load of concurrent update operations where the interval between updates is small.

MCMS can also handle a single continuous data feed such as one coming from a live news source. However, MCMS performance will suffer where a live source is coupled with authored content updates.

Design-time performance of MCMS is largely independent of the size of the database as shown below:

Site architecture and management

Site architecture can have considerable impact on performance. Site architecture includes the network configuration of the site and the design of the internal data structures that Content Management Server uses.



Dedicated hardware

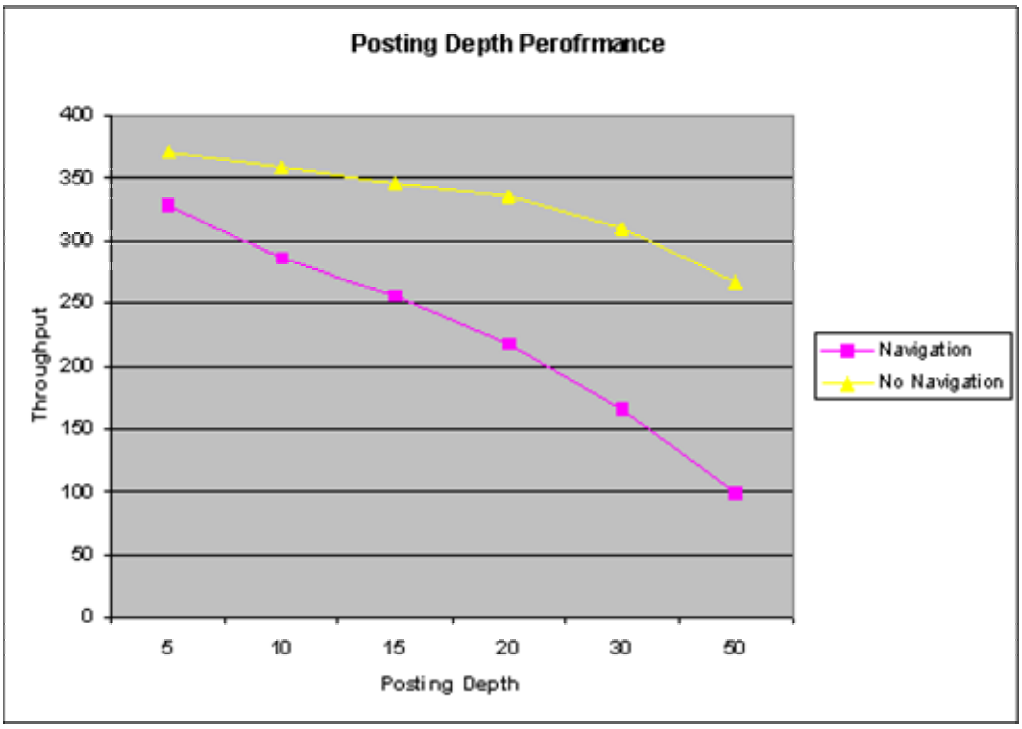
To avoid competition for resources among applications, it is important that you keep MCMS on dedicated servers. SQL Servers should be installed on their own machines as well. This will avoid the applications and services competing for CPU and memory.

Also be sure to perform transaction cost analysis (TCA) on your site before you deploy to fully understand the hardware and network bandwidth requirements for your deployment. See the white paper on Transaction Cost Analysis and Capacity Planning for MCMS for details.

Container hierarchy

There are two types of containers in the MCMS object model: channels and galleries. Channels contain pages and determine the site structure. Channels also allow you to control which users can view, edit, and approve those pages. Galleries contain templates or resources such as images. Galleries allow you to organize these objects in the database as you might files in a directory structure, and also allow you to control user access to these objects.

You can establish quite complex container hierarchies to capture both the site structure and the user rights required to control the site content. However, large container hierarchies carry their own performance cost. Therefore, it is best to limit the depth of container hierarchy as seen by results in the table below:



- **Root Channel containers:** Limit the number of containers under the root node. (For example, channels should not have more than 10 to 15 immediate children containers.)
- **Items in a container**: Limit the items in a container to less than 300. You can do this by distributing items over multiple containers, to ensure that the number of items in each container does not exceed 300.

Resource management

Resources such as images, sound clips, and video clips can have a large impact on your Web site's performance. As a general rule, the use of large resources should be avoided. Compress resources whenever possible. For example, for pictures, use .jpg images instead of .bmp images.

A managed resource is one that is stored in an MCMS resource gallery item and can be shared by all pages. An unmanaged resource is a resource that is referenced through a URL and may be part of the template itself.

Because unmanaged resources are not part of MCMS, you can achieve performance improvements by using them. However, this can mean a trade-off between manageability and performance.

Site partitioning

For the very largest sites, scaling up, scaling out, and following performance best practices may not be enough to deliver the desired performance. Once a site exceeds a certain size, it can no longer be cached effectively on a single MCMS server, and you may have to partition it into two or more sites.

Consider a very large hypothetical site with 20,000 frequently accessed pages. Depending on the size of the individual pages, an MCMS server that has already been scaled up to the maximum memory of 4 GB RAM might still be able to cache only 10,000 pages. This will leave 10,000 pages to be served from the disk cache or the database, slowing performance significantly. Scaling out by adding more MCMS servers to the cluster or more clusters to the site will not help, and the individual servers cannot be scaled up because they already have the maximum memory supported by Windows.

However, two smaller sites, each containing 10,000 of the most frequently accessed pages, will be able to keep all their pages in output or fragment cache, which will improve performance accordingly.

Best Practice: Very large MCMS sites that contain many postings should consider whether it is possible to partition the site into a series of smaller sites to get optimum performance. These smaller sites can each have their own MCMS database or can share a single MCMS database. Separate databases for each site can also deliver some performance gain when accessing content from the SQL Server as the database size of each individual site is smaller than the database size of the combined sites. However, the key performance benefits will come from maximizing the number of pages that can be stored in cache.

Site security

Secure Sockets Layer (SSL) is the security protocol used to secure Web sites. SSL is a connection-layer protocol. It works by establishing a secure connection between the client computer and the server and encrypting all data passing through that connection. This secure connection permits the transmission of sensitive data in both directions, and is often used for e-commerce sites and extranets which feature private information.

Because SSL connections require complex cryptographic functions to encrypt and decrypt data, their use will impact the performance of your application. The largest performance hit occurs during the initial connection, called the handshake, where asymmetric (public key) cryptography is used. Once the handshake establishes a shared session key between the server and the browser, faster symmetric encryption is used for bulk encryption of all data over the connection. Therefore, the performance cost includes both the large one-time cost of the handshake and the incremental extra cost to encrypt each message passed after the handshake.

To optimize pages that use SSL, reduce the amount of data that must be encrypted by using less text and simple graphics. If an SSL session lasts longer than the limit set by the server, it will time out and a new session will have to be established, complete with handshake. To avoid having to repeat the handshake, you should fine tune the session time by increasing the value of the `ServerCacheTime` registry entry. The session length

should be set to accommodate any reasonable session length, based on your user profile.

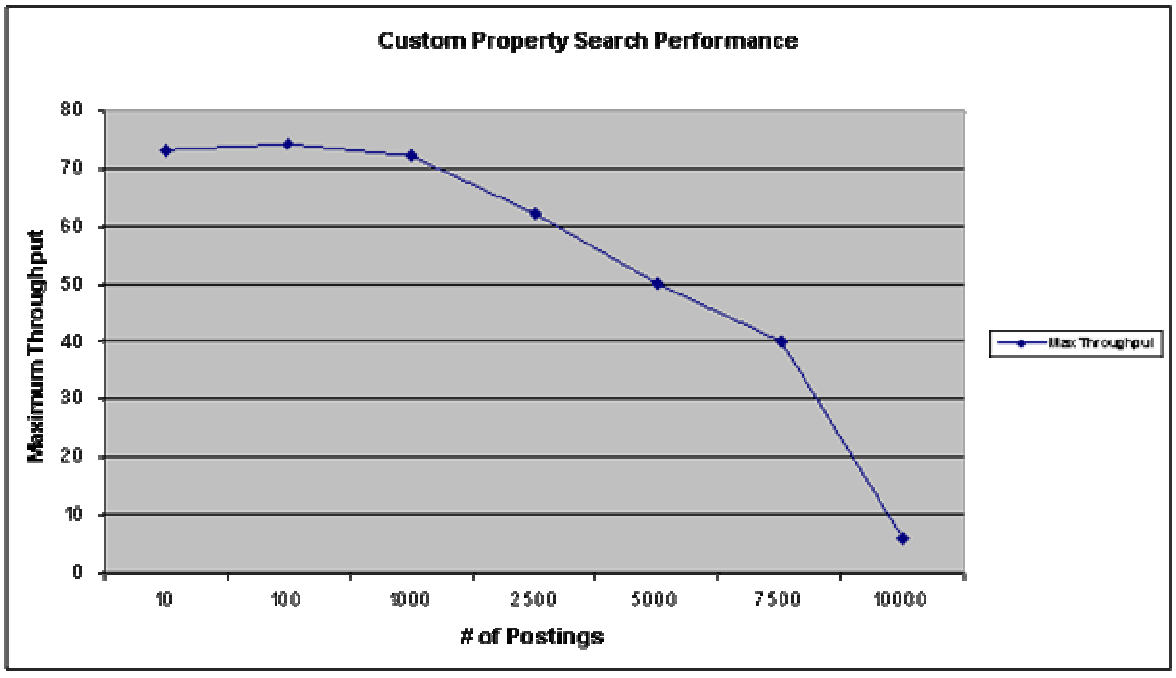
For more information, see article Q247658, "Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication" in the Microsoft Knowledge Base.

Custom property searches

Custom properties are user-defined properties that allow developers to define their own metadata for a page or channel. Custom properties allow site developers to create summaries of pages and channel content, cross reference pages to channels, and categorize pages and channels based on their content. By defining custom properties in templates and using them in the pages, site content can be managed programmatically in very sophisticated ways.

Custom property searches, of course, have a processing cost. As the chart below indicates, searches of low to moderate scope will not significantly impact performance. However, as the number of postings searched exceeds 500, the processing cost on the MCMS server results in a decline in throughput. As the number of postings searched reaches a critical point, the search will begin to affect the SQL database as well. While this point will vary depending on the structure of your site, testing indicates that searches of 5000 or more postings will create a bottleneck at the database as the CPU capacity of the SQL server is reached. Performance will, of course, suffer greatly.

To prevent custom property searches from impacting performance, therefore, limit the number searches performed and their scope in terms of total postings searched.



Troubleshooting

Careful planning and design can eliminate most performance bottlenecks before they ever occur. However, you may find during site testing or in actual production that performance issues arise, and you may have to fine-tune your site. This troubleshooting section outlines some tips and tools that will help to diagnose the problem and suggest potential fixes.

Testing your site

Site testing is an important step that can identify performance issues before your Web site goes live. To get the best results, follow the guidelines below. Your test environment should resemble your production environment as closely as possible. This ensures that the data you collect from the test environment is accurate.

Create and measure test performance

Creating and measuring test performance involves three stages. First, determine the site usage profile. Second, use the usage profile data to create a test script using the Microsoft Application Center Test (ACT) or Web Application Stress (WAS) tool. Third, measure throughput and response time of the site.

Create a usage profile of the site. When measuring and testing site performance, you must consider what the business is trying to achieve, user behavior on the site, and the number of site visitors who will be served. These considerations will help you understand the usage profile of your site. Create a list of transactions that will be performed on the site and the frequency with which they will be performed so you can measure the site performance baseline.

For more information about usage profiles, see "Creating a Usage Profile for Site Capacity Planning" **http://go.microsoft.com/fwlink/?LinkId=9508** . This article provides instructions for creating a usage profile.

Use the usage profile data to create a test script by using the ACT tool. The ACT tool is used to test the stress and endurance of the site. During stress testing, a high load is placed on the server for a period of time. A high load is defined as one that uses 85 to 90 percent of the server capacity. A normal load is defined as one that uses 50 percent of the server capacity. In endurance testing, normal load levels are run over an extended period of time.

Measure the throughput and response time of the site. Throughput is an important measurement in identifying performance bottlenecks and improving system performance. Throughput refers to the number of client requests processed within a certain unit of time. Typically, the unit of measurement is requests per second or pages per second.

For more information about throughput, see "Understanding Performance Testing" **http://go.microsoft.com/fwlink/?LinkId=9511**.

Transaction Cost Analysis

You can use Transaction Cost Analysis (TCA) methodology to aid in capacity planning and to detect performance bottlenecks. For more information about TCA concepts and

methodologies, see the white paper "Transaction Cost Analysis and Capacity Planning for MCMS 2002".

Performance counters

Content Management Server 2002 has a series of performance counters that can provide certain information about how your MCMS Web site is functioning. Individual counters can be indicators of specific issues but generally they must be used in conjunction with other counters or MCMS information in order to remedy the problem.

The following is the complete list of performance counters provided in MCMS 2002:

Counter	Counts	Description
Active Enterprise (AE) Node objects	Number of active MCMS COM objects on the server	These are the COM nodes created as each request is processed. Each request will generate many COM objects in the AE Server object, which are then destroyed as the processing completes. This counter is a snapshot of the number of COM objects active at any one time and is a reflection of the amount of processing activity within the MCMS Server. In itself this does not provide a direct gauge of performance, however high values would demonstrate a very active MCMS site. If the site is underperforming this value can be decreased by using techniques such as caching the navigation controls, which in turn could improve performance.
AE Node objects created/sec	Number of AE node objects created per second	As above, but this provides the average activity over time.
Guest sessions	Number of guest sessions on the server	This is the number of current connections to the MCMS server that are authenticated as a Guest user. This is more for informational purposes than to adjust performance, although it could be used to identify peaks of activity.
Guest sessions opened/sec	Number of guest sessions opened per second	As above, but this provides the average activity over time.
Authenticated sessions	Number of authenticated sessions connected to server	This is the number of current authenticated connections to the MCMS server. This is more for informational purposes than to adjust performance, although it could be used to identify peaks of activity.

Authenticated sessions opened/sec	Number of authenticated sessions opened per second	As above, but this provides the average activity over time.
Edit sessions	Authoring or development sessions connected to server	This is the number of current authenticated connections to the MCMS server that are in Edit mode. This is more for informational purposes than to adjust performance, although it could be used to identify peaks of activity. Edit activity on an MCMS sever can have a significant impact on performance.
Edit sessions opened/sec	Number of authoring or development sessions opened per second	As above, but this provides the average activity over time.
ISAPI sessions	Number of Internet Server API (ISAPI) sessions opened by server	Provides the number of connections currently in the MCMS ISAPI filter performing URL transformations. If you have a high number of these, it may mean that the URL transformation is taking a long time, which may be adjusted by increasing the node cache size.
ISAPI sessions opened/sec	Number of ISAPI sessions opened per second	Tells you how many connections are opened per second in the MCMS ISAPI filter. Provides the number of URL requests to MCMS per second.
Master cache nodes	Number of items in internal MCMS master cache	Number of nodes in master cache. This is limited by the node cache size set in the SCA. If this value is less than the setting in the SCA then this is an indication that all requested MCMS nodes are currently cached. If this value is close to or above the value in the SCA, then this is an indication that the setting in the SCA is not high enough to allow all requested nodes to be cached.
Shared nodes	Number of items/nodes referenced by server, including master cache items	This value is related to the number of master cache nodes; however this value will reflect multiple versions, e.g. checked in and checked out, of the same master node.
Shared nodes	Shared nodes	This is a reflection of the increase in size of

created/sec	created per second	the node cache as new nodes are cached. If the master node cache value is close to or above the value in the SCA and this value is high, then this is a clear indication that objects are being moved in and out of cache which will lower performance.
Cache hits/sec	Rate of cache hits on master cache	A high value here relative to the value below implies an effective use of cache.
Cache misses/sec	Rate of cache misses on master cache	A high value here relative to the value above implies an inefficient cache setting.
Data access operations/sec	Number of data access operations executed per second	This is the count of accesses to the SQL Server database. A high value could imply one of several things: the node cache value may be too small; the resource cache may be too small; or there is a need to cache search results.
Exceptions thrown	Number of exceptions thrown by server	This is informational and provides a count of exceptions thrown. If this is increasing rapidly over time it could indicate problems on the server or in the code base.
Number of MCMS connections	Number of open MCMS application connections	This is informational and provides an indication of the load on the MCMS server. It includes all current connections to the MCMS server at a point in time. It includes the ISAPI connections plus all other connections.

Diagnosis

The performance counters can provide your first strong clues as to where the trouble lies. You can further narrow down the causes by experimenting with system variables or design choices. When doing so, it is important that you test one variable at a time. You can use the performance counters, system logs, event logs, and so forth to track the results of your changes.

The first step in diagnosing performance issues is to confirm whether it is a MCMS issue on an external issue. For example, if you have another ISAPI filter that preprocesses requests before MCMS receives them, measure the latency of both.

Once you have determined that the bottleneck occurs after the request reaches MCMS, examine the internal and external placeholders. For example, if there are placeholders that are Web services or pointers to external content or applications, examine the latency of those applications independently and then in conjunction with MCMS.

When you run into issues that can be traced back to the template, be sure to further investigate the problem by examining one variable at a time. For instance, there are

usually several controls on a page. If you cannot identify the source by inspecting the code, test the template by removing one control at a time. You may need to make adjustments to account for any connected contents, but this procedure will give you a good general idea of which control is the root cause of the performance issue.

Server

If the server seems slow in response or is rejecting requests, review the performance counters and check the number of connections being made and the current number of connections.

If you do not detect any trouble there, you may be experiencing slowdown due to tables being locked in the SQL database. This could be occurring due to site deployment (SD) or background processing. Check the scheduling for background processing or check for SD logs being created. It is important to schedule these at low site volume times.

Troubleshooting Matrix

The table below lists a number of issues and their possible causes and solutions.

Behavior	Solution
Rate of cache misses is high	Examine size and balance of caches. Ensure that output cache directives are correct and the durations are appropriate.
Rate of cache misses is high + data access operations are high	Adjust your cache settings and usage. This could also be the result of large searches—try caching the most common searches.
Rate of cache misses is high + number of master node cache items is high	Increase the size of the master node cache—ideally your peak node cache requirements should be around 80% of allocated node cache size to allow for shared nodes. You can also use this in conjunction with the ASP.NET performance counters for the output cache to trace the caching of data before MCMS.
Number of exceptions thrown by server is high	Check the event log.

Also, examine the number of authenticated sessions versus the number of guest sessions to measure how much each of these costs. If the number of authenticated sessions has a large impact, examine site authentication protocol. See “Authentication” above for more details.

Test Platform

Throughout this white paper, charts and results have been presented based on testing done by Microsoft. This testing was done using the following test configuration, except in scale-up tests:

CMS Server	Compaq Proliant DL 380
	2 x PIII Processor @ 1200 MHz
	1 GB Memory
	Compaq NC3163 Fast Ethernet NIC
	Windows 2000 SP3 Advanced Server
	IISLockdown installed
	CMS using read-only mixed mode setup
Database Server	Compaq Proliant DL 580
	4 x PIII Xeon Processors @ 700 MHz
	1 GB Memory
	Intel PRO 10/100 Adapter
	Windows 2000 SP2 Advanced Server
	SQL 2000 SP3
Client Machines	Sufficient ACT Clients
Network	Switched Network on Cisco 2948G Fiber Switch
	Dual proc PIII AD DC

For more information:

http://www.microsoft.com/cmserver.